

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра Системотехніки

Звіт

з лабораторної роботи №4

з дисципліни: «Технології Високопродуктивних Обчислень»

з теми: «ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ  
OPENMP»

Виконав:

здобувач освіти першого  
(бакалаврського) рівня освіти гр.

КНТ-22-1

Орлов О. С.

Варіант: №9

Перевірів:

Професор кафедри СТ  
Міщеряков Ю. В.

## 4 ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ OPENMP

### 4.1 Мета роботи

Ознайомлення з принципами роботи паралельних програм із використанням технології OpenMP, засобами їх розробки і запуску.

### 4.2 Хід роботи

Згідно з варіантом №9, необхідно розробити програму для знаходження оберненої матриці методом Гаусса. Алгоритм був розпаралелений за допомогою директиви `#pragma omp parallel for`, яка розподіляє ітерації циклу між потоками.

Нижче наведено код розробленої програми мовою C:

```
#include <math.h>
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 1000

void initialize_matrix(double **matrix) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < 2 * N; j++) {
            if (j < N) {
                matrix[i][j] = (rand() % 100) + 1;
            } else {
                matrix[i][j] = (j == (i + N)) ? 1.0 : 0.0;
            }
        }
    }
}

void print_matrix(double **matrix) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < 2 * N; j++) {
            printf("%.2f ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main(int argc, char **argv) {
    srand(time(NULL));

    double **matrix = (double **)malloc(N * sizeof(double *));
    for (int i = 0; i < N; i++) {
        matrix[i] = (double *)malloc(2 * N * sizeof(double));
    }

    initialize_matrix(matrix);
```

```

printf("Matrix size: %dx%d\n", N, N);
// print_matrix(matrix);

double start_time = omp_get_wtime();

for (int i = 0; i < N; i++) {
    double pivot = matrix[i][i];

    if (fabs(pivot) < 1e-7) {
        puts("Matrix is singular, cannot find inverted, exiting.");
        exit(1);
    }

    for (int j = 0; j < 2 * N; j++) {
        matrix[i][j] /= pivot;
    }

#pragma omp parallel for
    for (int k = 0; k < N; k++) {
        if (k != i) {
            double factor = matrix[k][i];
            for (int j = 0; j < 2 * N; j++) {
                matrix[k][j] -= factor * matrix[i][j];
            }
        }
    }
}

double end_time = omp_get_wtime();
double time_taken = end_time - start_time;
// printf("\n");
// print_matrix(matrix);

int threads = omp_get_max_threads();

printf("Threads: %d\n", threads);
printf("Time taken: %f seconds\n", time_taken);

for (int i = 0; i < N; i++) {
    free(matrix[i]);
}
free(matrix);

return 0;
}

```

### 4.3 Результати роботи

Проведено серію запусків програми зі змінною кількістю потоків (1, 2, 4, 8, 16) для матриці розмірністю  $1000 \times 1000$ .

```
dxrkness@pc ~/u/t/lab4> OMP_NUM_THREADS=1 ./main  
Matrix size: 1000x1000  
Threads: 1  
Time taken: 8.833628 seconds
```

Рисунок 4.1 – Виконання для одного потоку

```
dxrkness@pc ~/u/t/lab4> OMP_NUM_THREADS=2 ./main  
Matrix size: 1000x1000  
Threads: 2  
Time taken: 5.019141 seconds
```

Рисунок 4.2 – Виконання для двох потоків

```
dxrkness@pc ~/u/t/lab4> OMP_NUM_THREADS=4 ./main  
Matrix size: 1000x1000  
Threads: 4  
Time taken: 2.617860 seconds
```

Рисунок 4.3 – Виконання для чотирьох потоків

```
dxrkness@pc ~/u/t/lab4> OMP_NUM_THREADS=8 ./main  
Matrix size: 1000x1000  
Threads: 8  
Time taken: 1.628530 seconds
```

Рисунок 4.4 – Виконання для вісьмох потоків

```
dxrkness@pc ~/u/t/lab4> OMP_NUM_THREADS=16 ./main  
Matrix size: 1000x1000  
Threads: 16  
Time taken: 3.282355 seconds
```

Рисунок 4.5 – Виконання для шістнадцяти потоків

Результати вимірювань часу виконання та розрахунок прискорення занесено у таблицю:

Таблиця 4.1 – Показники ефективності паралельного алгоритму

Кількість потоків	Час виконання (с)	Прискорення ( $S$ )	Ефективність ( $E$ )
1	8.83	1.00	100%
2	5.02	1.76	88%
4	2.62	3.37	84%
8	1.63	5.42	68%
16	3.28	2.69	17%

#### 4.4 Аналіз результатів

Найбільше прискорення спостерігається при переході з 4 на 8 потоків, що зумовлено наявністю в комп'ютері восьми логічних потоків (чотирьох фізичних ядер). При переході на 16 потоків, час стрімко падає через постійний context switching.

#### 4.5 Висновки

У ході лабораторної роботи було вивчено технологію OpenMP та принципи розробки багатопотокових додатків для систем зі спільною пам'яттю. Реалізовано паралельний алгоритм знаходження оберненої матриці методом Гаусса. Практичні результати показали, що використання OpenMP дозволяє суттєво зменшити час виконання обчислювально складних задач. Однак, прискорення не є лінійним через накладні витрати на керування потоками та доступ до пам'яті.